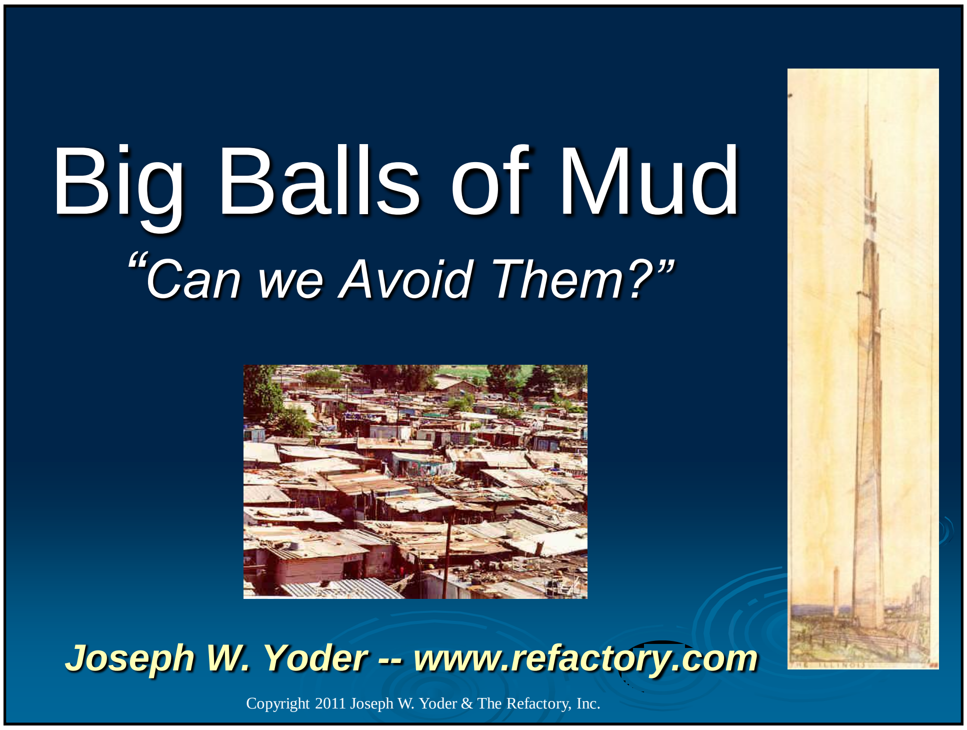
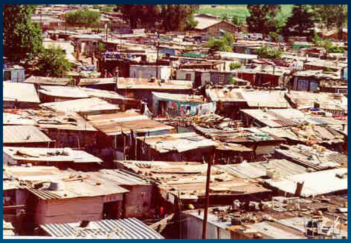
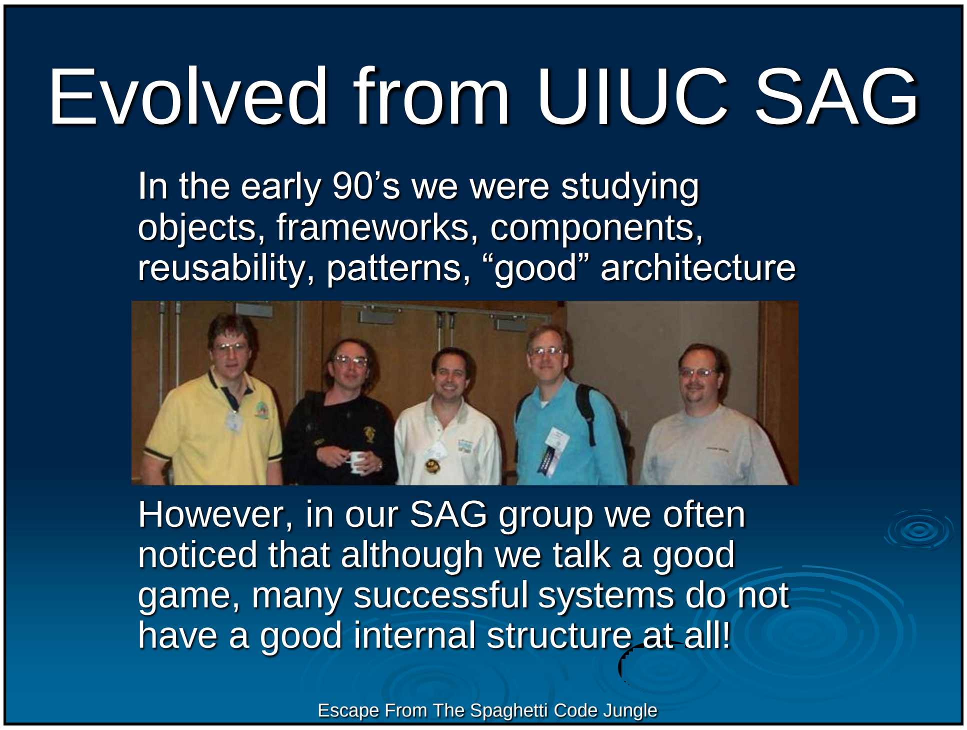


Big Balls of Mud

"Can we Avoid Them?"



Joseph W. Yoder -- www.refactory.com



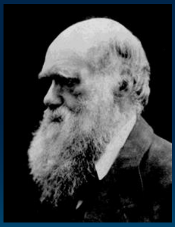
Evolved from UIUC SAG

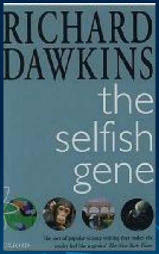
In the early 90's we were studying objects, frameworks, components, reusability, patterns, "good" architecture

However, in our SAG group we often noticed that although we talk a good game, many successful systems do not have a good internal structure at all!

Selfish Class

Brian and I had just published a paper called Selfish Class which takes a *code's-eye view of software reuse and evolution.*



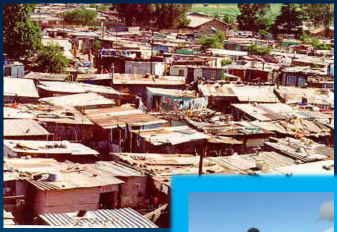


In contrast, our BBoM paper noted that in reality, a lot of code was hard to (re)-use.

Big Ball of Mud

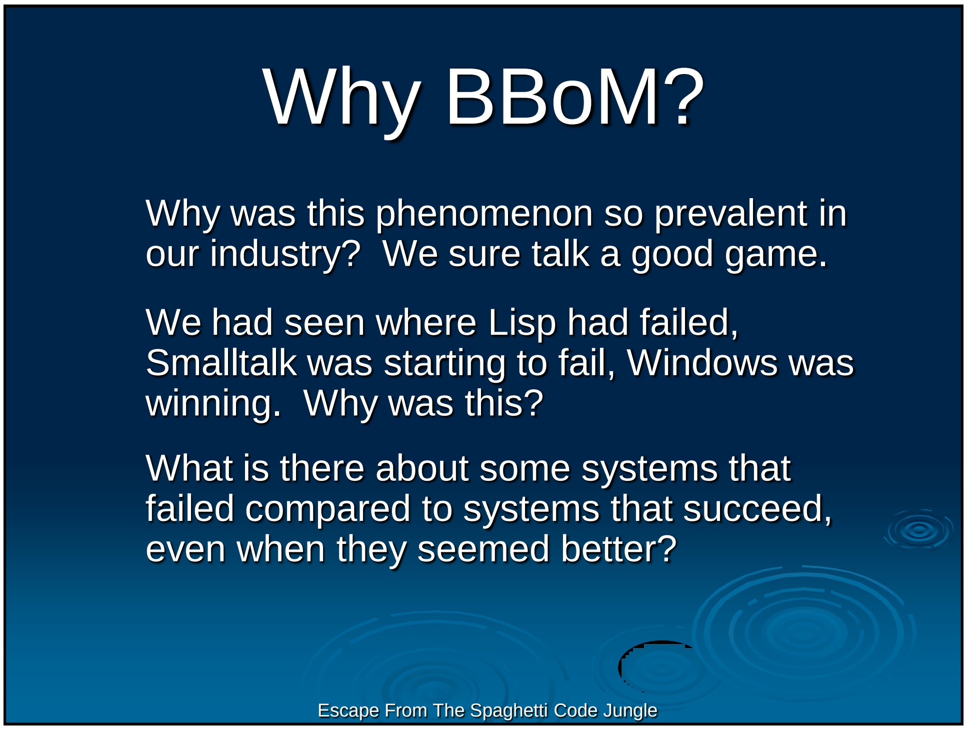
Alias: Shantytown, Spaghetti Code

A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle.



The de-facto standard software architecture. Why is the gap between what we **preach** and what we **practice** so large?

We preach we want to build high quality systems but why are BBoMs so prevalent?



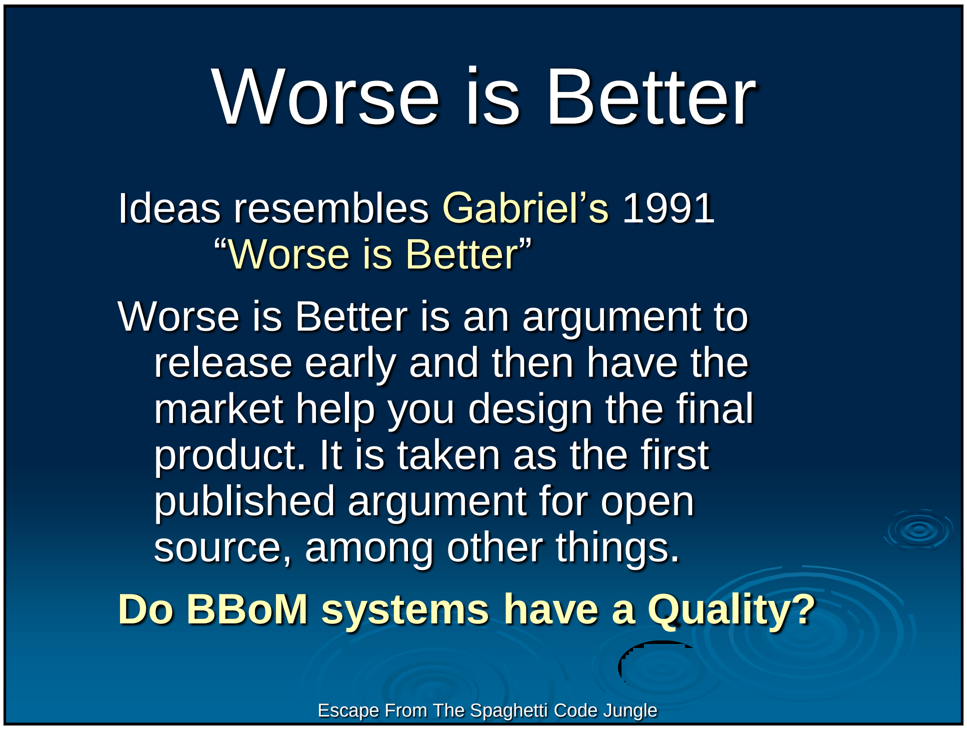
Why BBoM?

Why was this phenomenon so prevalent in our industry? We sure talk a good game.

We had seen where Lisp had failed, Smalltalk was starting to fail, Windows was winning. Why was this?

What is there about some systems that failed compared to systems that succeed, even when they seemed better?

Escape From The Spaghetti Code Jungle



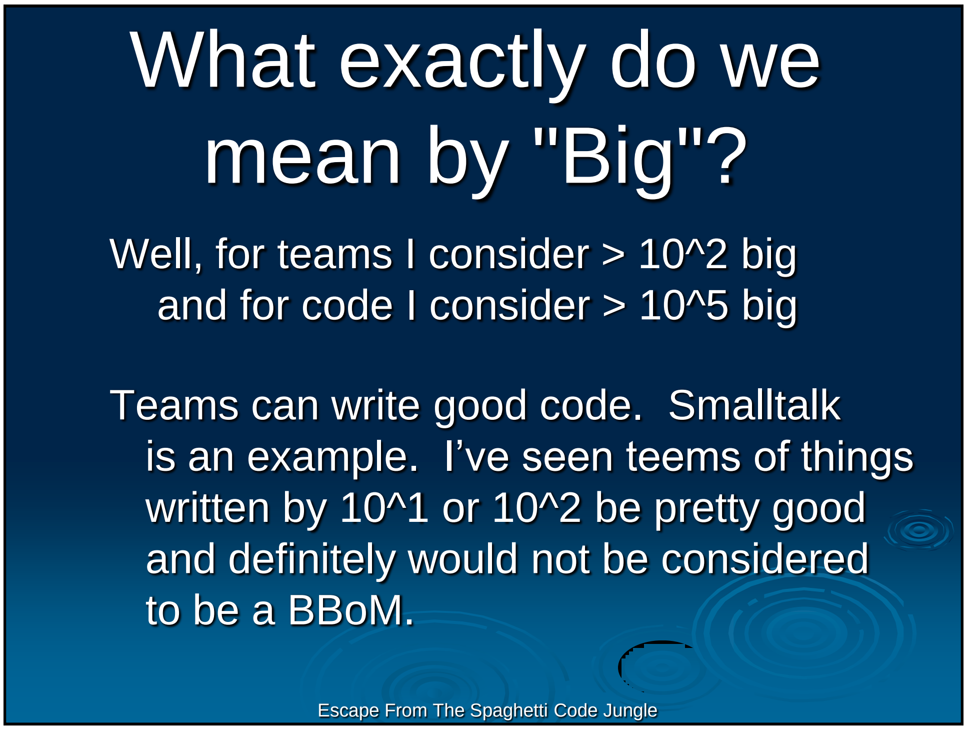
Worse is Better

Ideas resembles Gabriel's 1991 "Worse is Better"

Worse is Better is an argument to release early and then have the market help you design the final product. It is taken as the first published argument for open source, among other things.

Do BBoM systems have a Quality?

Escape From The Spaghetti Code Jungle

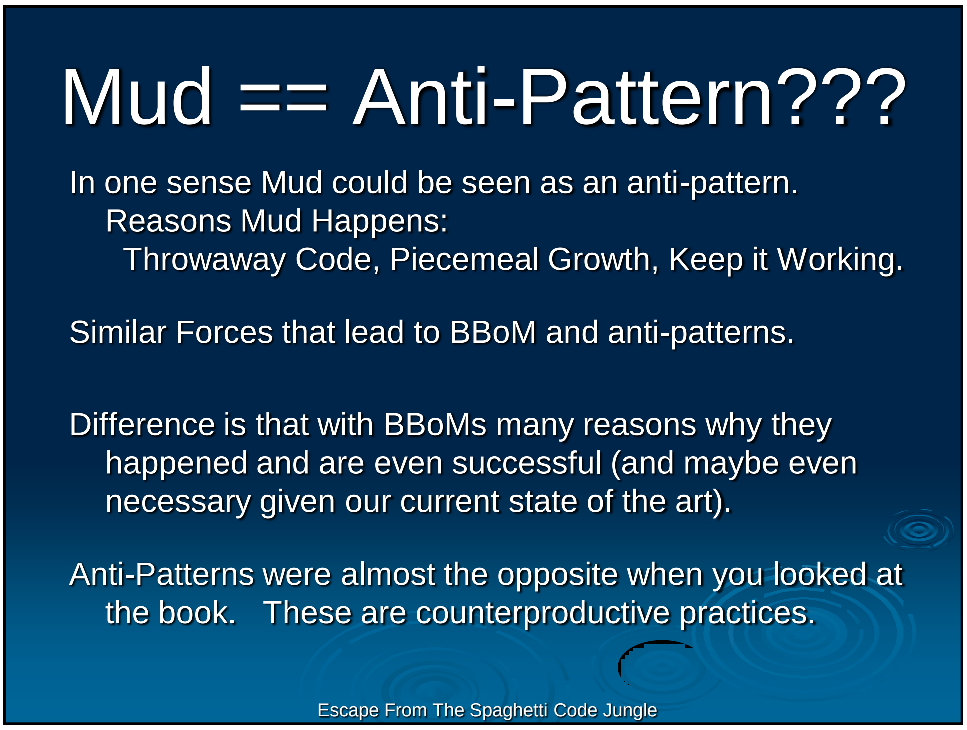


What exactly do we mean by "Big"?

Well, for teams I consider > 10^2 big
and for code I consider > 10^5 big

Teams can write good code. Smalltalk is an example. I’ve seen teems of things written by 10^1 or 10^2 be pretty good and definitely would not be considered to be a BBoM.

Escape From The Spaghetti Code Jungle



Mud == Anti-Pattern???

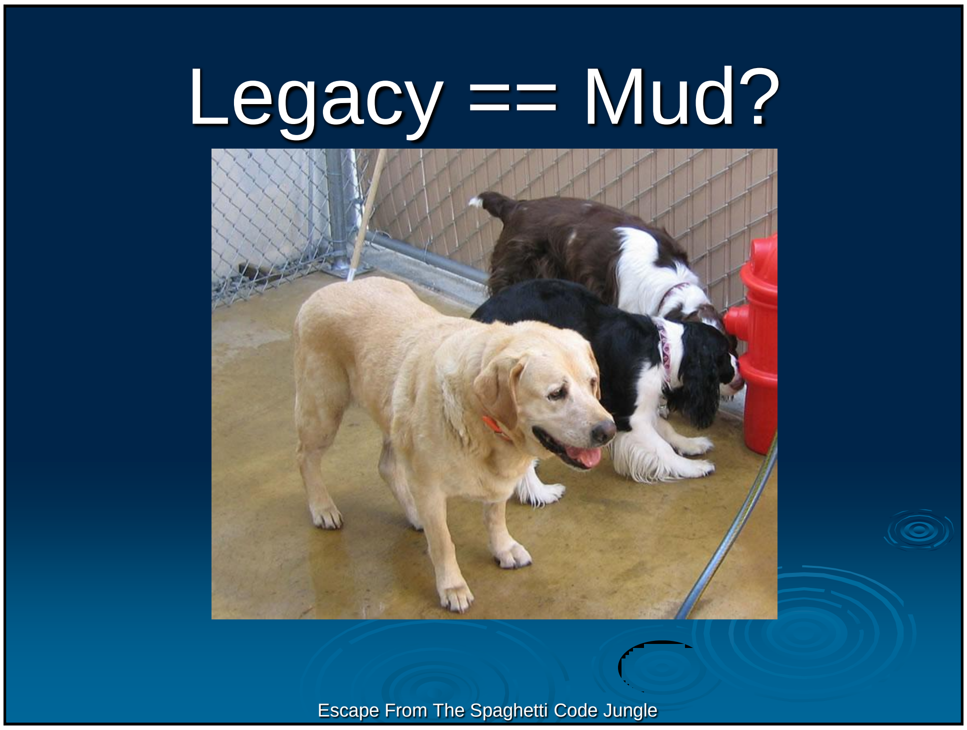
In one sense Mud could be seen as an anti-pattern.
Reasons Mud Happens:
Throwaway Code, Piecemeal Growth, Keep it Working.

Similar Forces that lead to BBoM and anti-patterns.

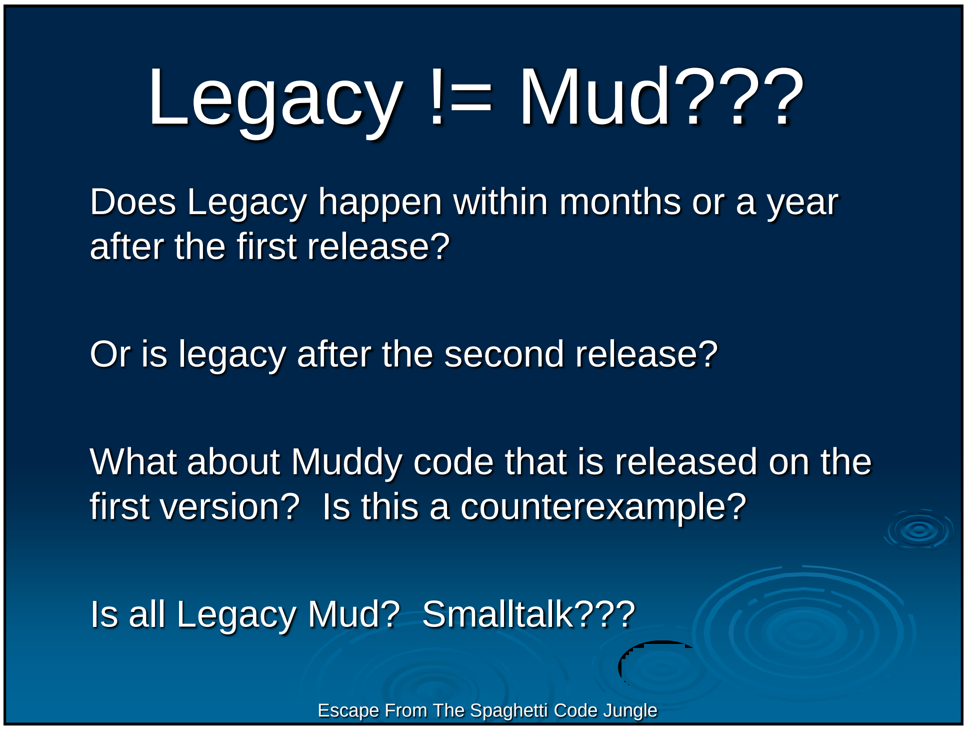
Difference is that with BBoMs many reasons why they happened and are even successful (and maybe even necessary given our current state of the art).

Anti-Patterns were almost the opposite when you looked at the book. These are counterproductive practices.

Escape From The Spaghetti Code Jungle



Legacy == Mud?



Legacy != Mud???

Does Legacy happen within months or a year after the first release?

Or is legacy after the second release?

What about Muddy code that is released on the first version? Is this a counterexample?

Is all Legacy Mud? Smalltalk???

Is Mud Normal?

Well, just read our paper....there are "normal" reasons why it happens. Maybe it is the best we can do right now.

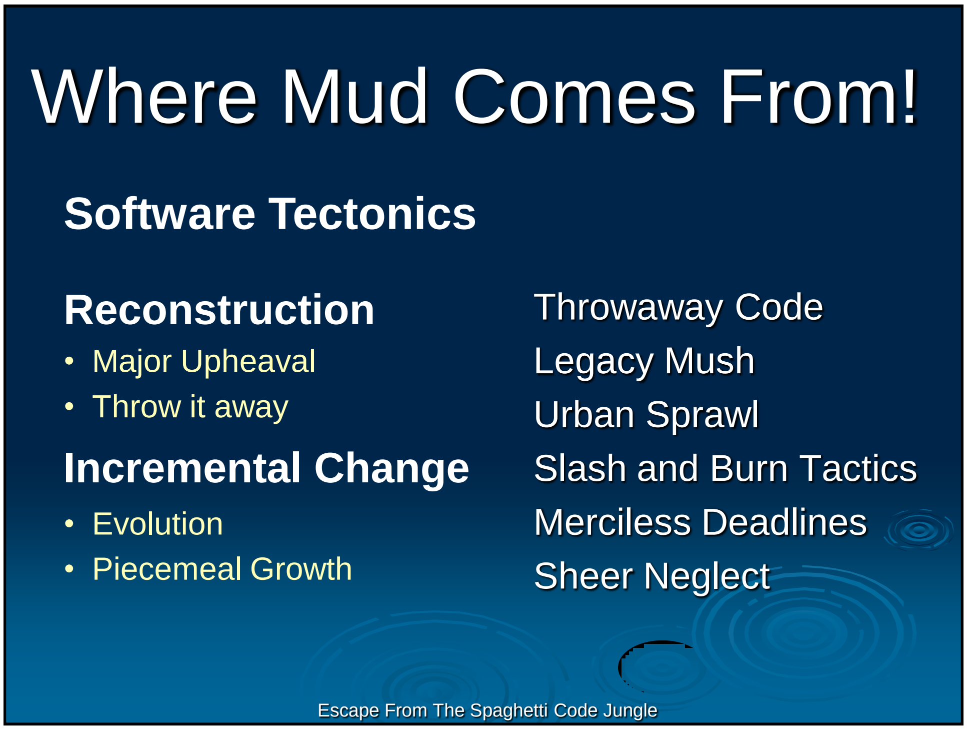
If mud is such a bad thing, why do people keep making it?

Maybe if we accept it and teach it more then we can deal with it better and help prevent it from getting too bad.

Where Mud Comes From?

HOW DO WE KNOW YOU CAME TO ELBONIA JUST TO TEACH US CAPITALISM ?
YEAH... MAYBE YOU CAME TO STEAL OUR SECRET PROCESS FOR MAKING MUD !!
DIRT AND WATER ?
HE KNOWS...
WE'LL HAVE TO KILL HIM.
© 1990 United Feature Syndicate, Inc.
DILBERT © United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited.

People Write Code → People make Mud



Where Mud Comes From!

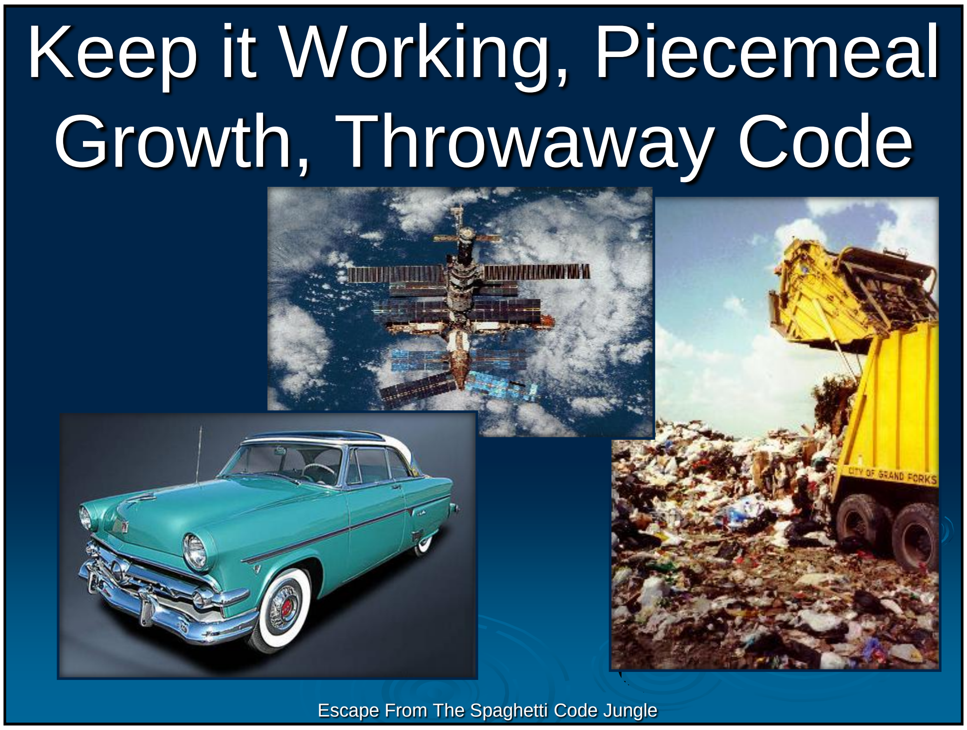
Software Tectonics

Reconstruction
- Major Upheaval
- Throw it away

Incremental Change
- Evolution
- Piecemeal Growth

Throwaway Code
Legacy Mush
Urban Sprawl
Slash and Burn Tactics
Merciless Deadlines
Sheer Neglect

Escape From The Spaghetti Code Jungle



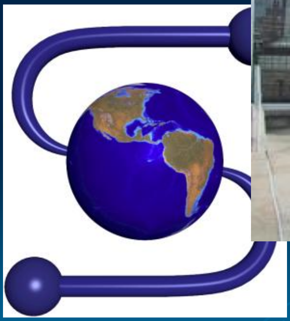
Keep it Working, Piecemeal Growth, Throwaway Code

Escape From The Spaghetti Code Jungle

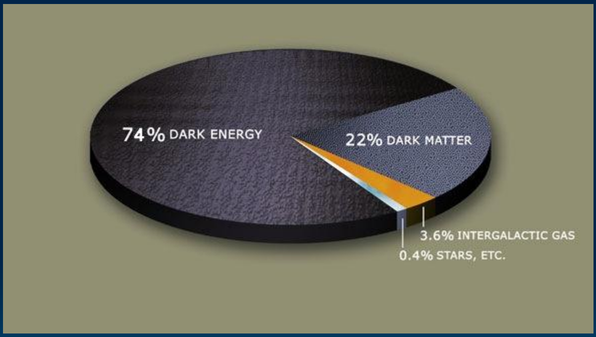
Copy 'n' Paste

The Age of Sampling

Big Bucket of Glue



The Mystery of Dark Matter



Accidental Complexity??? Maybe our current state of the art leads to Mud!

They Have a Name

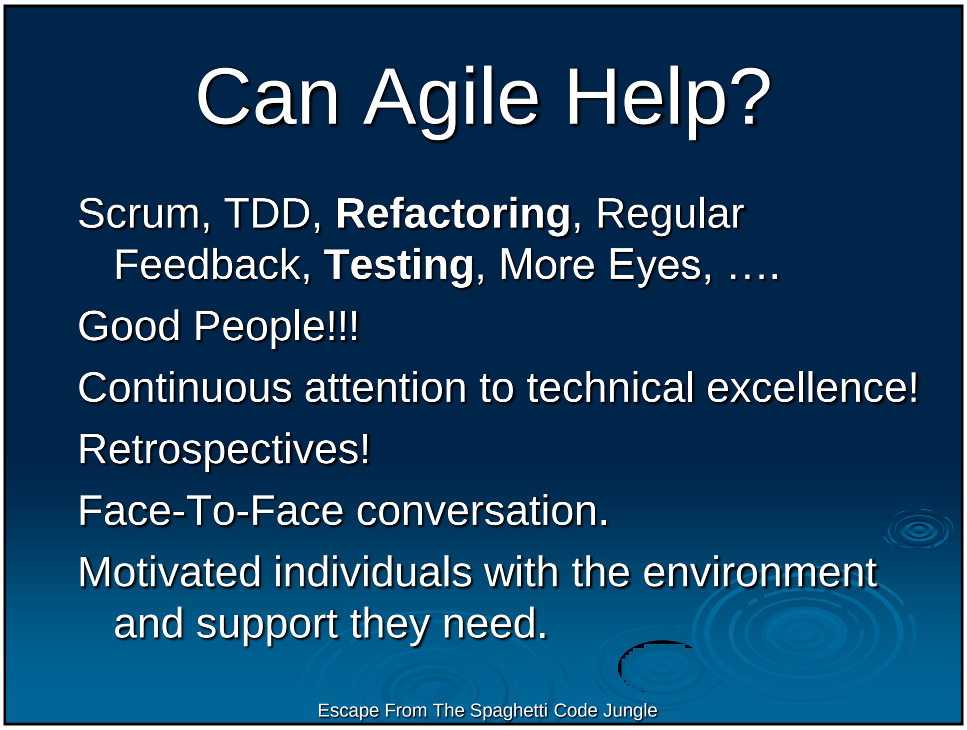
Millionaires / Billionaires

Agile to the Rescue???

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

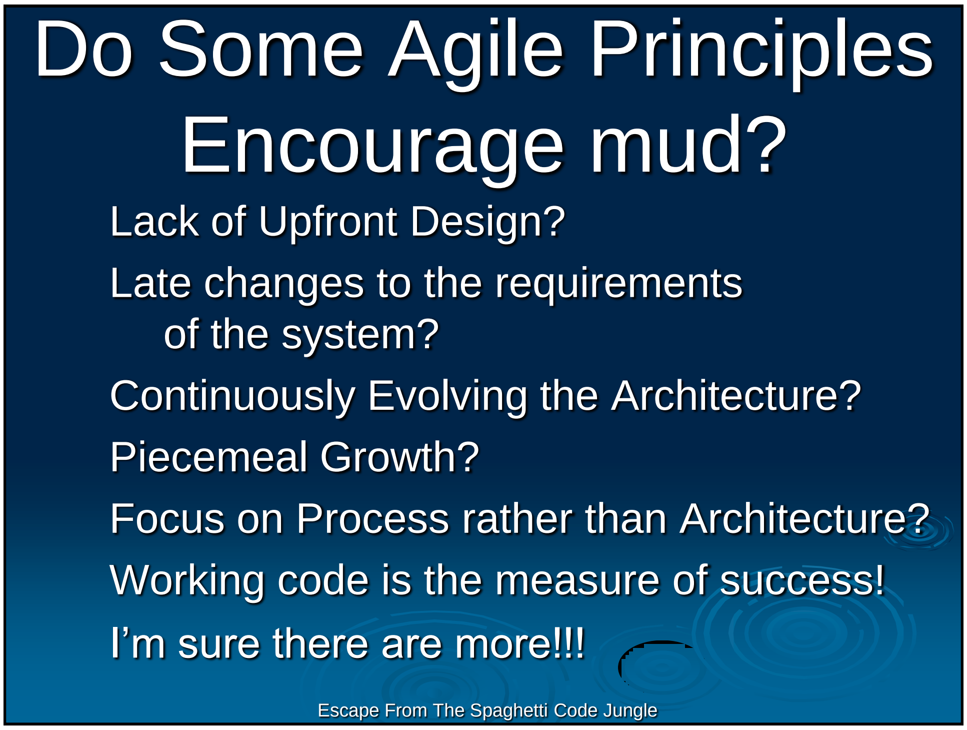
…From the Agile Manifesto



Can Agile Help?

- Scrum, TDD, **Refactoring**, Regular Feedback, **Testing**, More Eyes, ….
- Good People!!!
- Continuous attention to technical excellence!
- Retrospectives!
- Face-To-Face conversation.
- Motivated individuals with the environment and support they need.

Escape From The Spaghetti Code Jungle

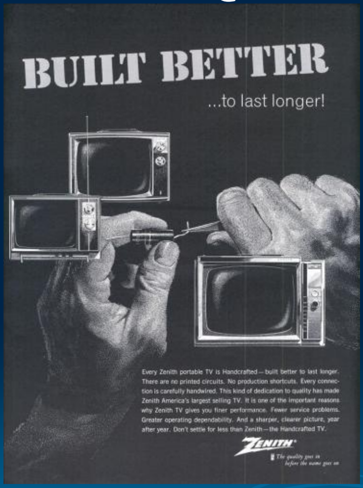


Do Some Agile Principles Encourage mud?

- Lack of Upfront Design?
- Late changes to the requirements of the system?
- Continuously Evolving the Architecture?
- Piecemeal Growth?
- Focus on Process rather than Architecture?
- Working code is the measure of success!
- I'm sure there are more!!!

Escape From The Spaghetti Code Jungle

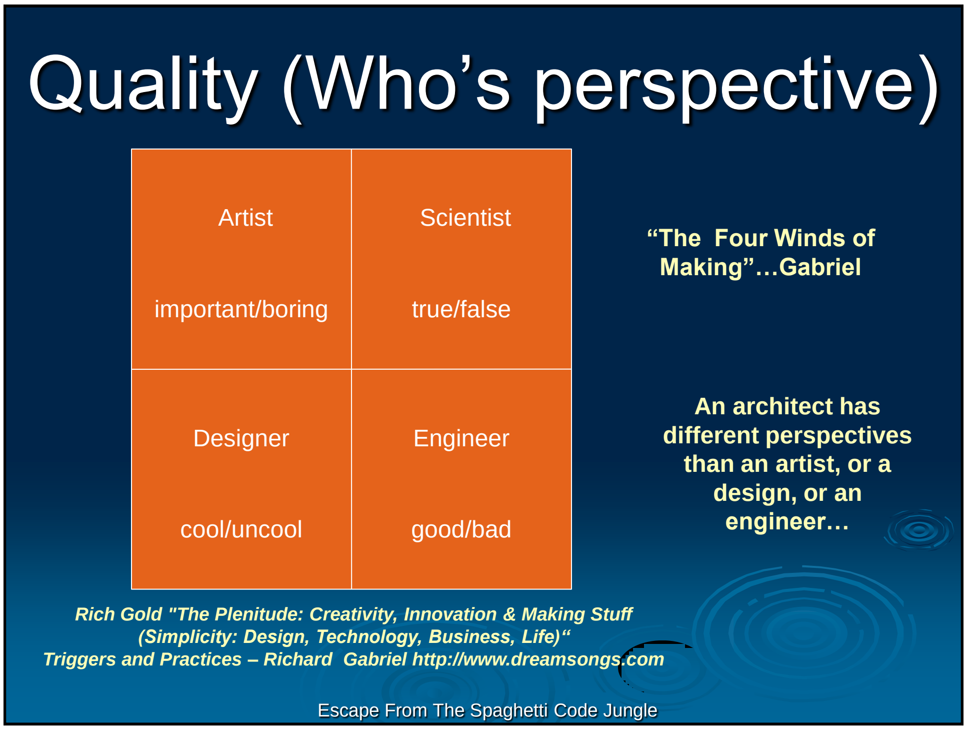
The Quality Goes In



Every Zenith portable TV is Handcrafted—built better to last longer. There are no printed circuits. No production shortcuts. Every connection is carefully handwired. This kind of dedication to quality has made Zenith America's largest selling TV. It is one of the important reasons why Zenith TV gives you finer performance. Fewer service problems. Greater operating dependability. And a sharper, clearer picture, year after year. Don't settle for less than Zenith—the Handcrafted TV.

Quality

Quality Definition: a peculiar and essential character or nature, an inherent feature or property, a degree of excellence or grade, a distinguishing attribute or characteristic



Quality (Who's perspective)

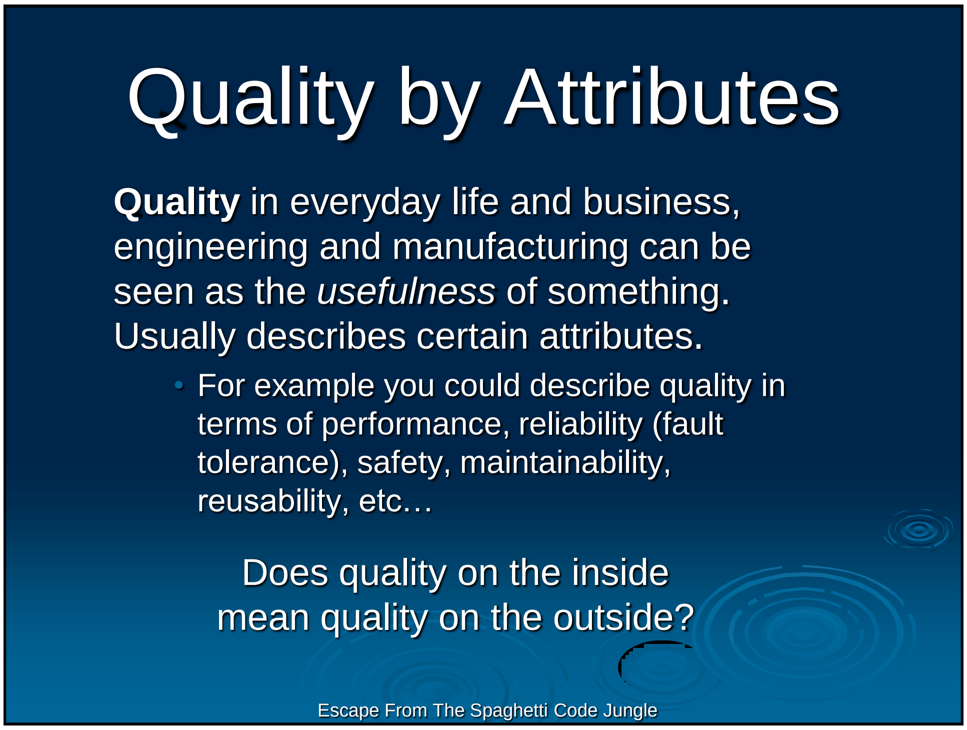
Artist important/boring	Scientist true/false
Designer cool/uncool	Engineer good/bad

"The Four Winds of Making"…Gabriel

An architect has different perspectives than an artist, or a design, or an engineer…

Rich Gold "The Plenitude: Creativity, Innovation & Making Stuff (Simplicity: Design, Technology, Business, Life)"
Triggers and Practices – Richard Gabriel http://www.dreamsongs.com

Escape From The Spaghetti Code Jungle



Quality by Attributes

Quality in everyday life and business, engineering and manufacturing can be seen as the *usefulness* of something. Usually describes certain attributes.

- For example you could describe quality in terms of performance, reliability (fault tolerance), safety, maintainability, reusability, etc…

Does quality on the inside mean quality on the outside?

Escape From The Spaghetti Code Jungle

Non-functional Requirements

Accessibility	Reliability
Compatibility	Safety
Efficiency	Scalability
Effectiveness	Security
Extensibility	Stability
Maintainability	Supportability
Performance	Usability

Other terms for non-functional requirements are "constraints", "quality attributes“, and "quality of service requirements"

Qualities are usually described by “ilities” as seen in non-functional requirements…but quality can also focus on how well the functional requirements are met (how to measure this?)

Brooklyn Bridge

The **Brooklyn Bridge,** 1883, one of the oldest suspension bridges in the United States, stretches over a mile from Brooklyn to Manhattan.

On completion, it was the largest suspension bridge in the world and the first steel-wire suspension bridge.

Brooklyn Bridge

Over engineered. Had 6 times what it needed which proved useful over time

What happens if we tried overdesign of our systems (a language for printing hello world) or the same line of code 6 times (is this 6 times more reliable?)

```
if (x != a) x = a; if (x != a) x = a; if (x != a) x = a;
if (x != a) x = a; if (x != a) x = a; if (x != a) x = a;
```

Redundant components can make our systems more reliable

Being Good Enough

- Quality of being good enough.
- Does it meet the minimum requirements
- Quality has many competing forces…are we designing a system for online orders or for controlling the space shuttle, they have different qualities, thus different patterns and solutions apply.
- Perfection is the enemy of ***Good Enough!***
- Maybe Quality without a Number.

Problems with Quality

Too many people try to **Quantify** it. **Quality** is hard to **quantify**, or we'd call it **Quantity**. What makes a book good, a movie, a painting, ... should we measure the length of the Godfather II, and make all our movies that long, and achieve the same quality?

Hemmingway was one of Literary Engineering's leading figure, so we should study his sentence and paragraph lengths, and ensure our sentences and paragraphs conform to these Hemmingway metrics?

Metrics can help a little when it comes to quality but miss the point. **QWAN** can't really be quantify.

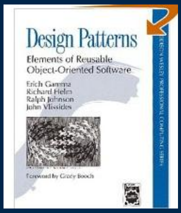
How about: "Quality Without A Number...(QWANum)"

Escape From The Spaghetti Code Jungle

Patterns and Quality

In a sense, patterns are all about quality

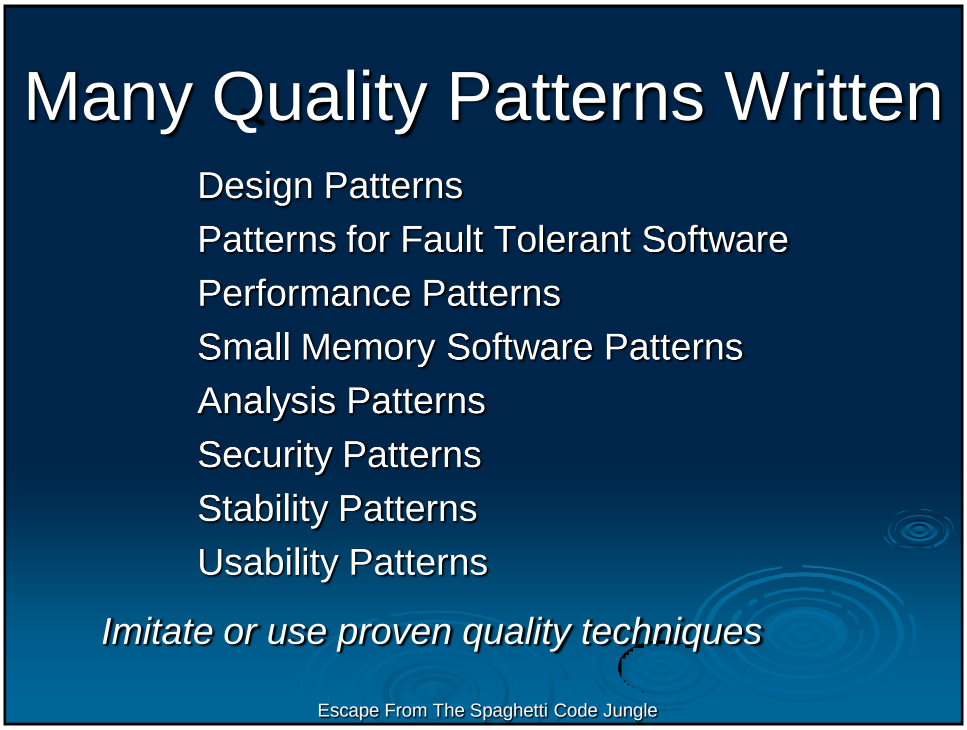
Design Patterns were about giving software the "quality" of being more reusable and maintainable… Quality of a good OO Design


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch

Fault Tolerance Patterns are about proven practices that help ensure build systems with the "quality" of being able to handle faults

PATTERNS FOR FAULT TOLERANT SOFTWARE

Escape From The Spaghetti Code Jungle

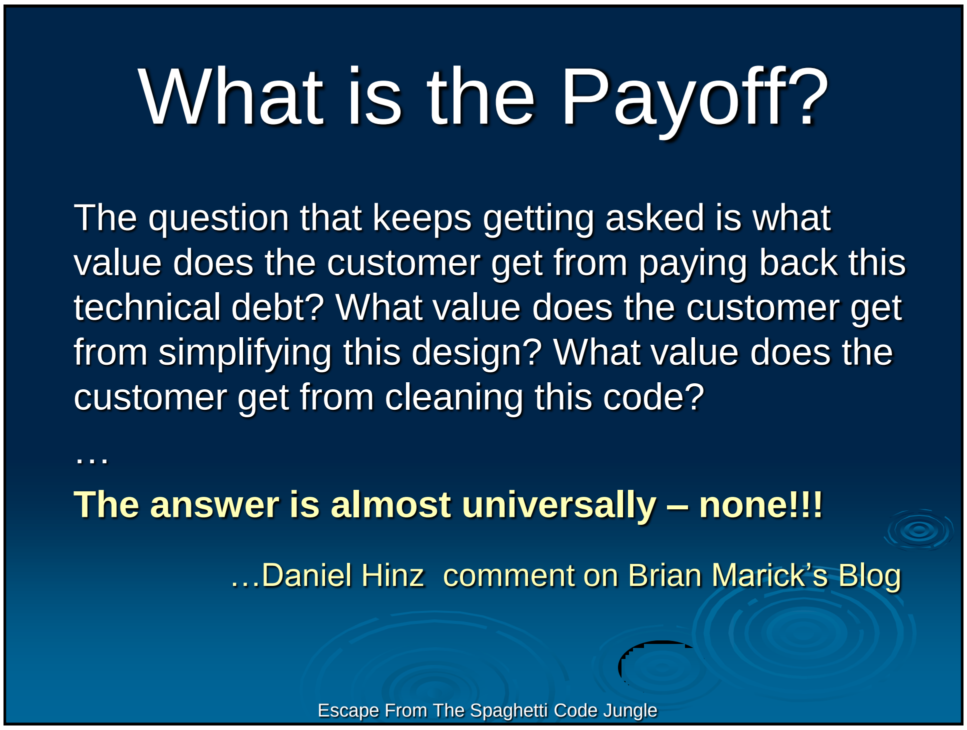


Many Quality Patterns Written

Design Patterns
Patterns for Fault Tolerant Software
Performance Patterns
Small Memory Software Patterns
Analysis Patterns
Security Patterns
Stability Patterns
Usability Patterns

Imitate or use proven quality techniques

Escape From The Spaghetti Code Jungle



What is the Payoff?

The question that keeps getting asked is what value does the customer get from paying back this technical debt? What value does the customer get from simplifying this design? What value does the customer get from cleaning this code?

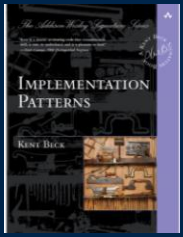
…

The answer is almost universally – none!!!

…Daniel Hinz comment on Brian Marick's Blog

Escape From The Spaghetti Code Jungle

Does Quality Code Matter?



Patterns about creating quality code that communicates well, is easy to understand, and is a pleasure to read. Book is about patterns of "Quality" code.

But…Kent states, "…*this book is built on a fragile premise: that good code matters. I've seen too much ugly code make too much money to believe that quality of code is either necessary or sufficient for commercial success or widespread use. However I still believe quality of code matters*."

Patterns assist with making code more bug free and easier to maintain and extend.

Some Answers to Mud!?!

Can we gentrify, rehabilitate, or make-over code helping clean up the mud?

Can **refactoring**, patterns, frameworks, components, agile, and objects help with mud?

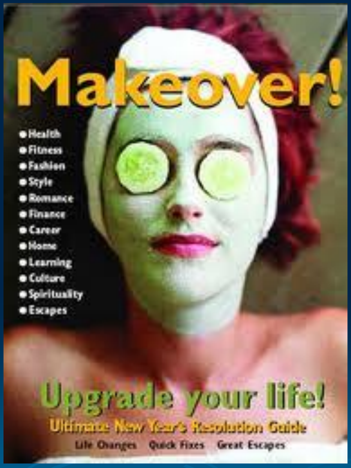
Total Code Makeover

Can we just Refactor out of Mud?

Sweep the Mess Under the Rug?

Total Code Makeover

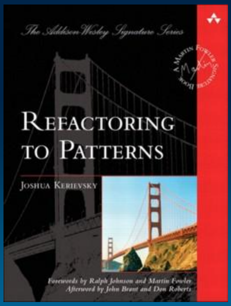
US WEEKLY
BRIT'S DRUG TEST DRAMA!
KANYE'S MOM Tragic Death By Plastic Surgery
20 BEST MAKEOVERS
NO SURGERY! The secret ways stars go from ho-hum to HOT

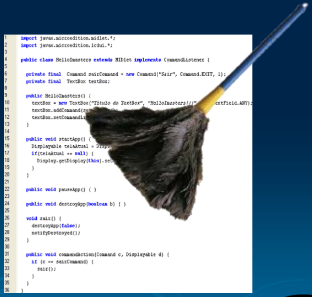


Code Make Over

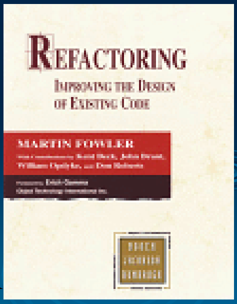
Refactoring can help reverse some mud. The tradeoff is cost and time....maybe with technology

Refactoring to Better Design (Patterns)…


The Addison-Wesley Signature Series
Refactoring to Patterns
Joshua Kerievsky
A Martin Fowler Signature Book
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts



```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloImasters extends MIDlet implements CommandListener {

  private final  Command sairCommand = new Command("Sair", Command.EXIT, 1);
  private final  TextBox textBox;

  public HelloImasters() {
    textBox = new TextBox("Titulo do TextBox", "HelloImasters!!!"      extField.ANY);
    textBox.addCommand(s
    textBox.setCommandLi
  }

  public void startApp() {
    Displayable telaAtual = Disp
    if(telaAtual == null) {
      Display.getDisplay(this).set
    }
  }

  public void pauseApp() { }

  public void destroyApp(boolean b) { }

  void sair() {
    destroyApp(false);
    notifyDestroyed();
  }

  public void commandAction(Command c, Displayable d) {
    if (c == sairCommand) {
      sair();
    }
  }
}
```


Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.

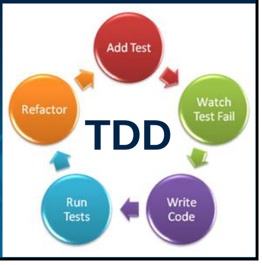
Refactorings

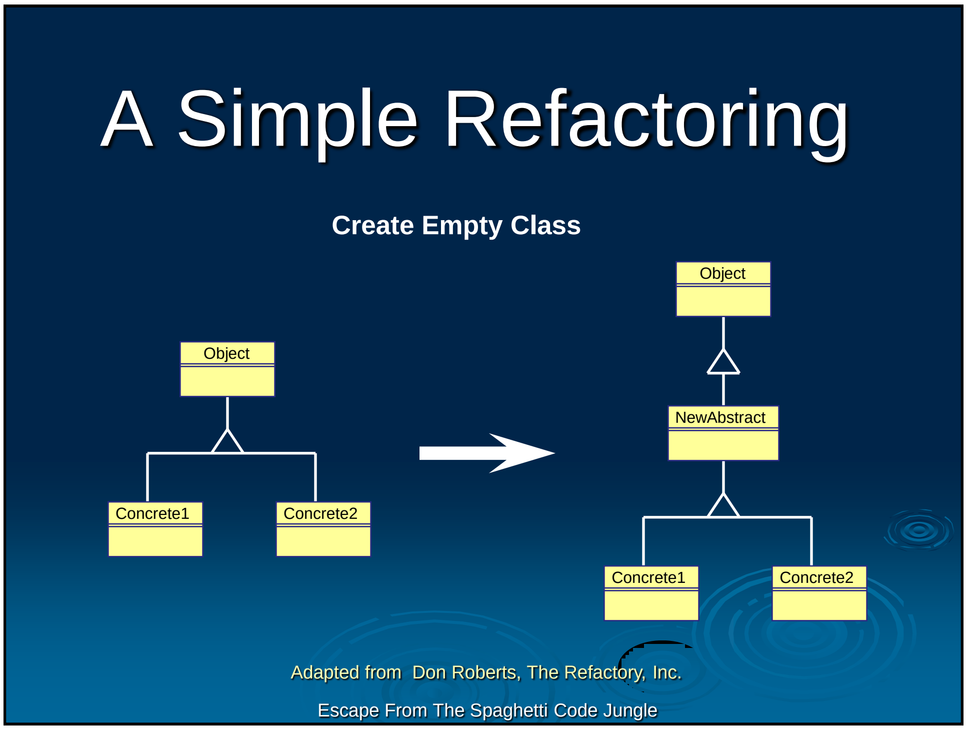
Behavior Preserving Program Transformations

- Rename Instance Variable
- Promote Method to Superclass
- Move Method to Component

Always done for a reason!!!

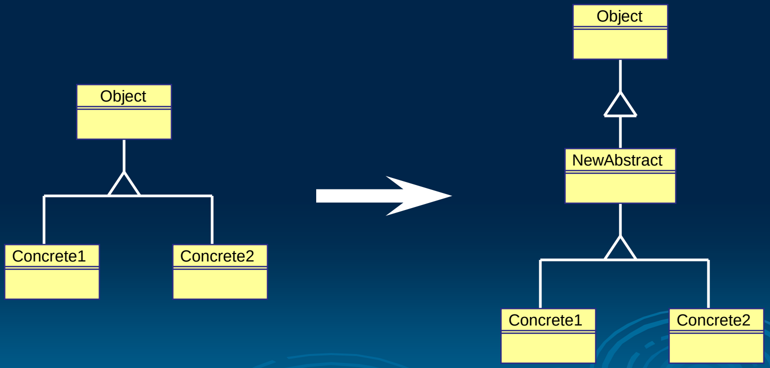
Refactoring is key and integral to most Agile processes!!!



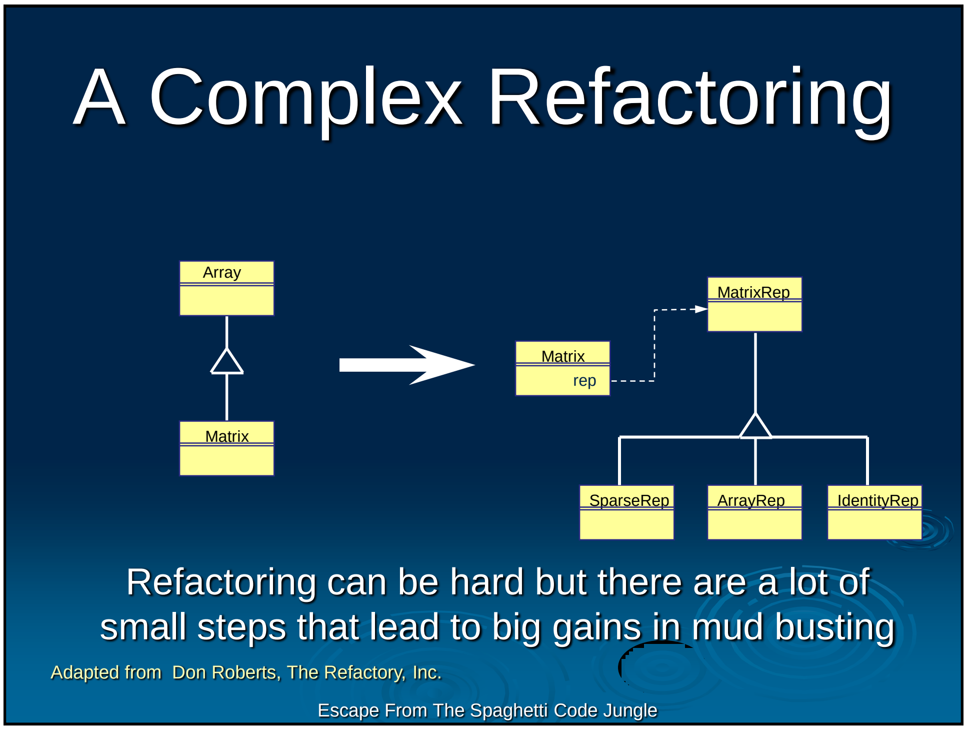


A Simple Refactoring

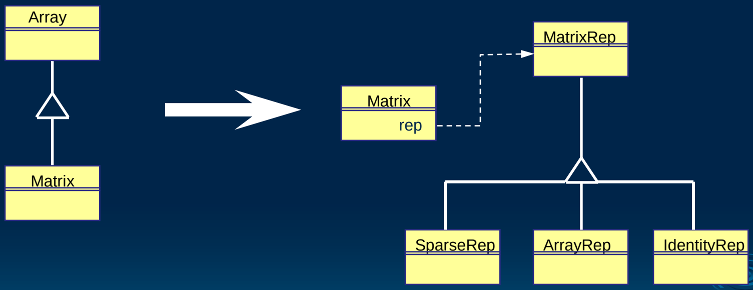
Create Empty Class



Adapted from Don Roberts, The Refactory, Inc.

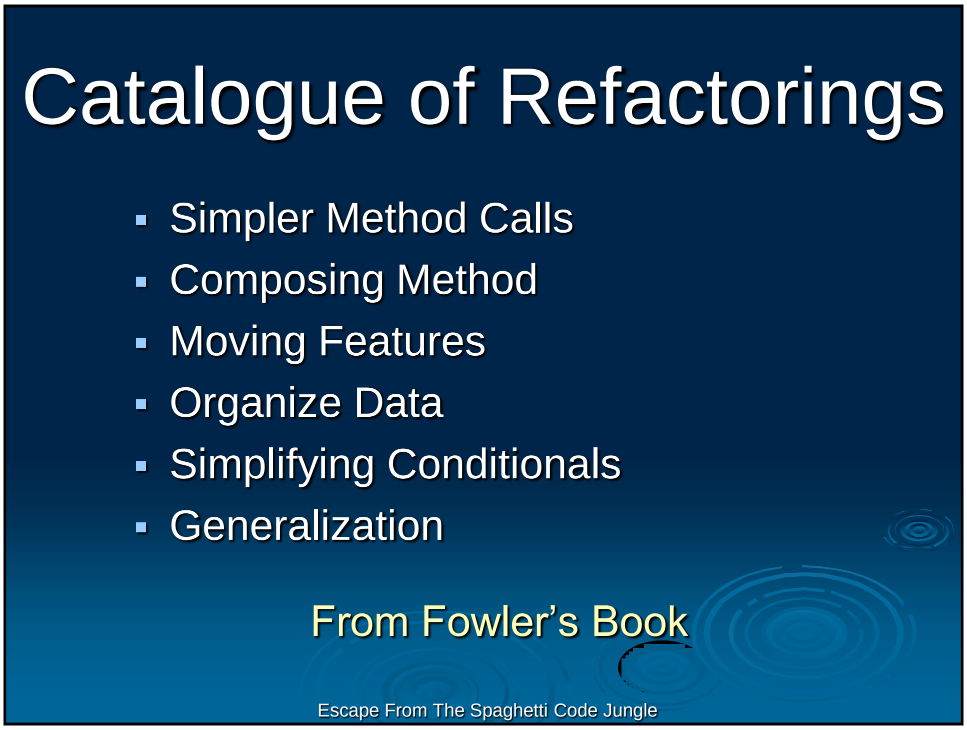


A Complex Refactoring



Refactoring can be hard but there are a lot of small steps that lead to big gains in mud busting

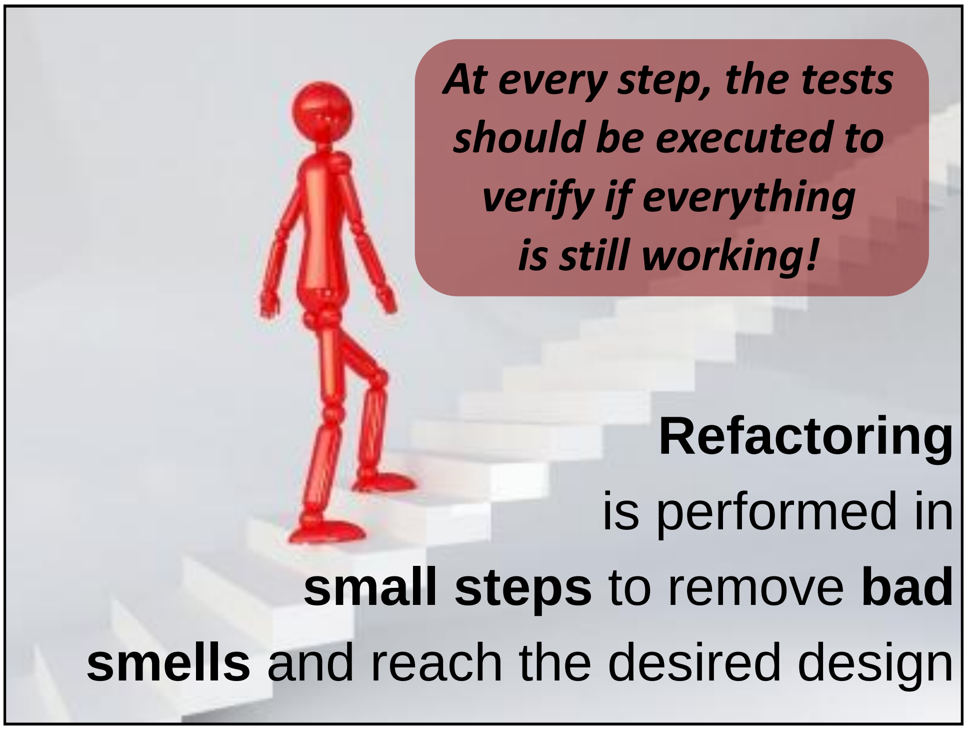
Adapted from Don Roberts, The Refactory, Inc.



Catalogue of Refactorings

- Simpler Method Calls
- Composing Method
- Moving Features
- Organize Data
- Simplifying Conditionals
- Generalization

From Fowler's Book



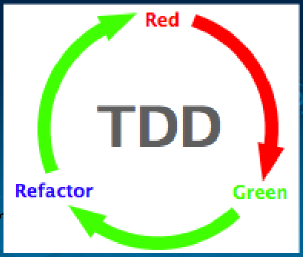
At every step, the tests should be executed to verify if everything is still working!

Refactoring is performed in **small steps** to remove **bad smells** and reach the desired design

Testing

You Must Test

- When you find smelly code, you often apply refactorings to clean your code.
- Testing is a key principle for safe refactoring!



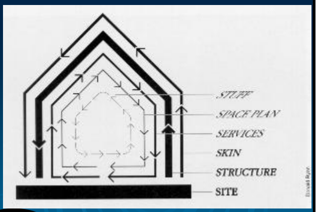
If we have a BBoM

How can we even start?

How can we cordon off the mess?

Stuart Brand's Shearing Layers

- Buildings are a set of components that evolve in different timescales.
- Layers: site, structure, skin, services, space plan, stuff. Each layer has its own value, and speed of change (pace).
- Buildings adapt because faster layers (services) are not obstructed by slower ones (structure).



—Stuart Brand, *How Buildings Learn*

Yoder and Foote's Software Shearing Layers

"Factor your system so that artifacts that change at similar rates are together."—Foote & Yoder, Ball of Mud, PLoPD4.

Slower

Layers

- The platform
- Infrastructure
- Data schema
- Standard frameworks and components
- Abstract classes and interfaces
- Classes
- Code
- Data

Faster

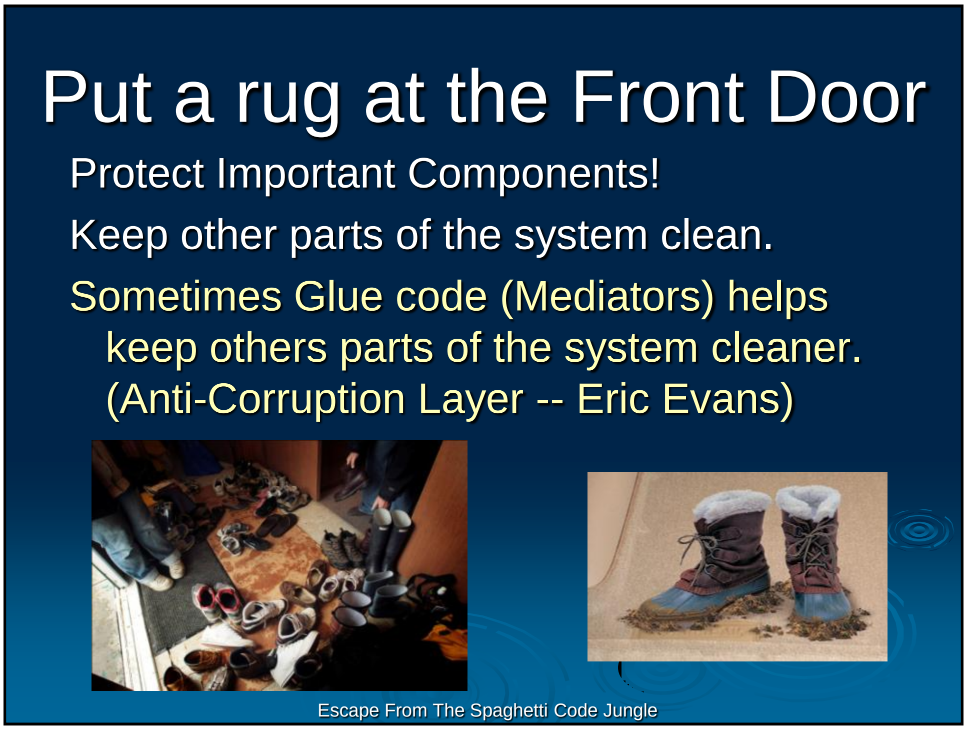
Escape From The Spaghetti Code Jungle

Sweep It Under the Rug

Allegations Against School Staff
CLAIM
CLAIM
We're going to need a bigger rug !

Cover it up to keep other areas clean (Façade and other Wrapper Patterns)

Escape From The Spaghetti Code Jungle

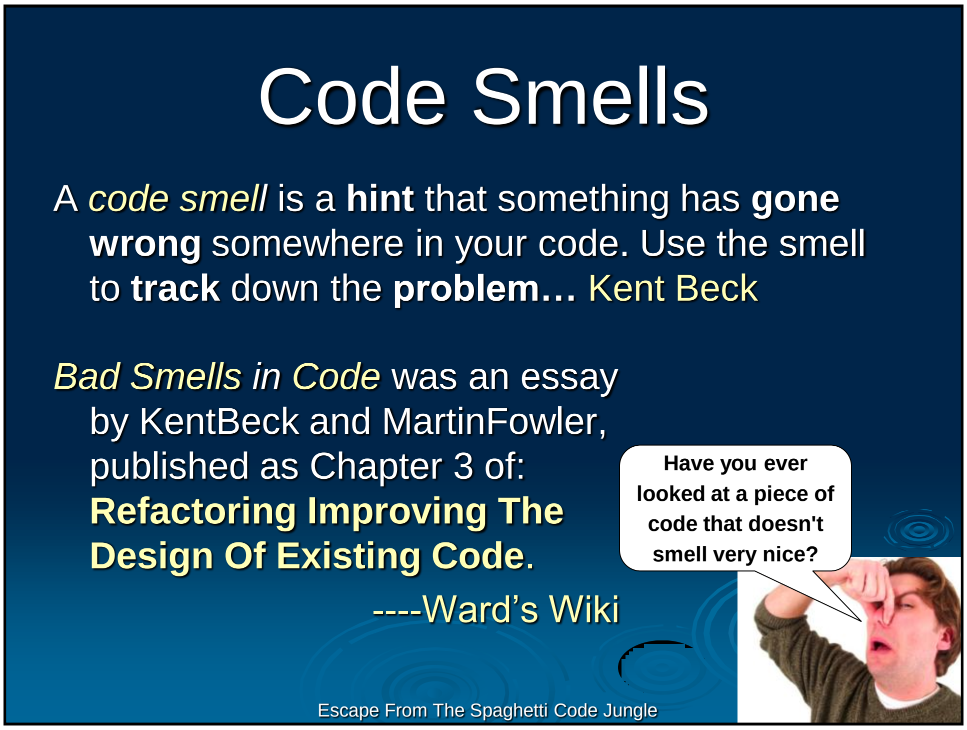


Put a rug at the Front Door

Protect Important Components!

Keep other parts of the system clean.

Sometimes Glue code (Mediators) helps keep others parts of the system cleaner. (Anti-Corruption Layer -- Eric Evans)



Code Smells

A *code smell* is a **hint** that something has **gone wrong** somewhere in your code. Use the smell to **track** down the **problem…** Kent Beck

Bad Smells in Code was an essay by KentBeck and MartinFowler, published as Chapter 3 of: **Refactoring Improving The Design Of Existing Code**.

----Ward's Wiki

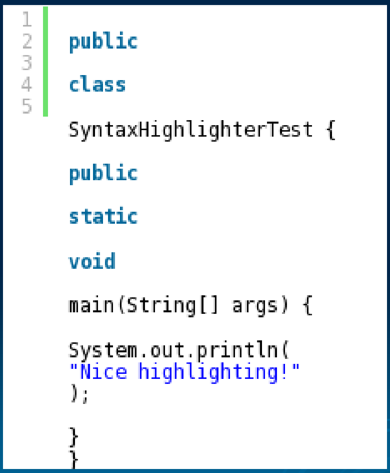
Have you ever looked at a piece of code that doesn't smell very nice?

Ten Most Putrid List

1) Sloppy Layout,
2) Dead Code,
3) Lame Names,
4) Commented Code,
5) Duplicated Code,
6) Feature Envy,
7) Inappropriate Intimacy,
8) Long Methods & Large Class,
9) Primitive Obsession & Long Parameter List,
10) Switch Statement & Conditional Complexity …

Escape From The Spaghetti Code Jungle

Bad Formatting



```
public

class

SyntaxHighlighterTest {

public

static

void

main(String[] args) {

System.out.println(
"Nice highlighting!"
);

}
}
```

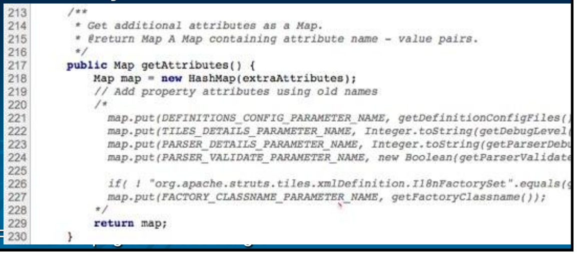
```
void foo(int x[], int y, int z){
if (z > y + 1)
{
int a = x[y], b = y + 1, c = z;
while (b < c)
{
if (x[b] <= a) b++; else {
            int d = x[b]; x[b] = x[--c];
     x[c] = d;
}
   }
int e = x[--b]; x[b] = x[y];
x[y] = e; foo(x, y, b);  bar(x, c, z);
}}

void bar(int i[], int j, int k)
{ return i[j] = int [k]}
```

Escape From The Spaghetti Code Jungle

Dead Code

```
void foo(int x[], int y, int z) {
  if (z > y + 1)  {
  int a = x[y], b = y + 1, c = z;
  while (b < c)  {
    if (x[b] <= a) b++; else {
      int d = x[b]; x[b] = x[--c];
      return;
      x[c] = d;
    }
    x[b] = a;
   }
  y = 5; // set y equal to 5
  int e = x[--b]; x[b] = x[y];
  x[y] = e; foo(x, y, b);
   /* used to use bar,
       might need it again
        bar(x, c, z); */
}
```

```
void bar(int i[], int j, int k) {
  /* bar method returning nothing */
  if (j > k) {
    return k
    i[k] = i[j];
  }
  if (j == k) {
    return i[j] = int [k]
  }
}
```



```
    /**
     * Get additional attributes as a Map.
     * @return Map A Map containing attribute name - value pairs.
     */
    public Map getAttributes() {
        Map map = new HashMap(extraAttributes);
        // Add property attributes using old names
        /*
          map.put(DEFINITIONS_CONFIG_PARAMETER_NAME, getDefinitionConfigFiles(
          map.put(TILES_DETAILS_PARAMETER_NAME, Integer.toString(getDebugLevel(
          map.put(PARSER_DETAILS_PARAMETER_NAME, Integer.toString(getParserDebu
          map.put(PARSER_VALIDATE_PARAMETER_NAME, new Boolean(getParserValidate

          if( ! "org.apache.struts.tiles.xmlDefinition.I18nFactorySet".equals(g
          map.put(FACTORY_CLASSNAME_PARAMETER_NAME, getFactoryClassname());
        */
        return map;
    }
```

Fix the Layout and Remove Useless Items

- Format the Code Consistently
  - Agree on a standard format
  - Set the tools for consistent formatting
  - Run the tools over the code base
- Remove Unreachable Code
  - Delete useless comments
  - Delete commented out code
  - Remove code that can't be reached,

Lame Names

```
void foo(int x[], int y, int z)
{
if (z > y + 1)
{
int a = x[y], b = y + 1, c = z;
while (b < c)
{
if (x[b] <= a) b++; else {
int d = x[b]; x[b] = x[--c];
x[c] = d;
}
}
int e = x[--b]; x[b] = x[y];
x[y] = e; foo(x, y, b);
foo(x, c, z);
}
```

```
void quicksort(int array[], int begin, int end) {
  if (end > begin + 1) {
    int pivot = array[begin],
    l = begin + 1, r = end;
    while (l < r)  {
      if (array[l] <= pivot)
        l++;
      else
      swap(&array[l], &array[--r]);
    }
    swap(&array[--l], &array[beg]);
    sort(array, begin, l);
    sort(array, r, end);
  }
}
```

http://dreamsongs.com/Files/BetterScienceThroughArt.pdf

Fixing Names

Names should *mean something.*

Standards improve communication
- know and follow them.

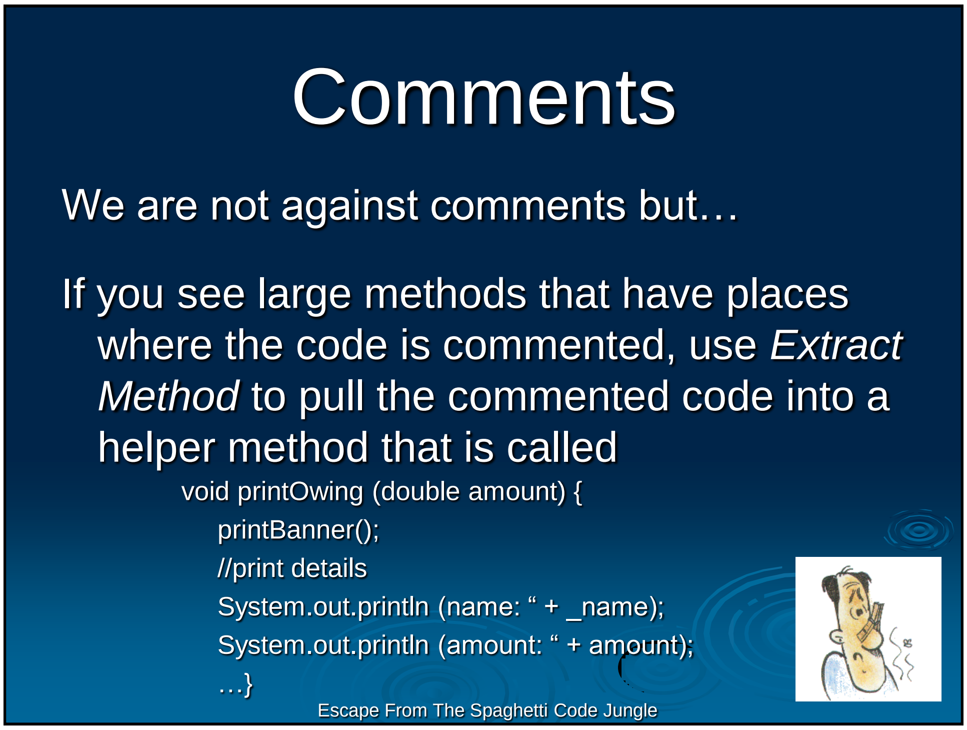
Standard protocols

object   ToString(), Equals()

ArrayList Contains(), Add(), AddRange()
Remove(), Count, RemoveAt(),

HashTable Keys, ContainsKey(),
ContainsValue()

Standard naming conventions

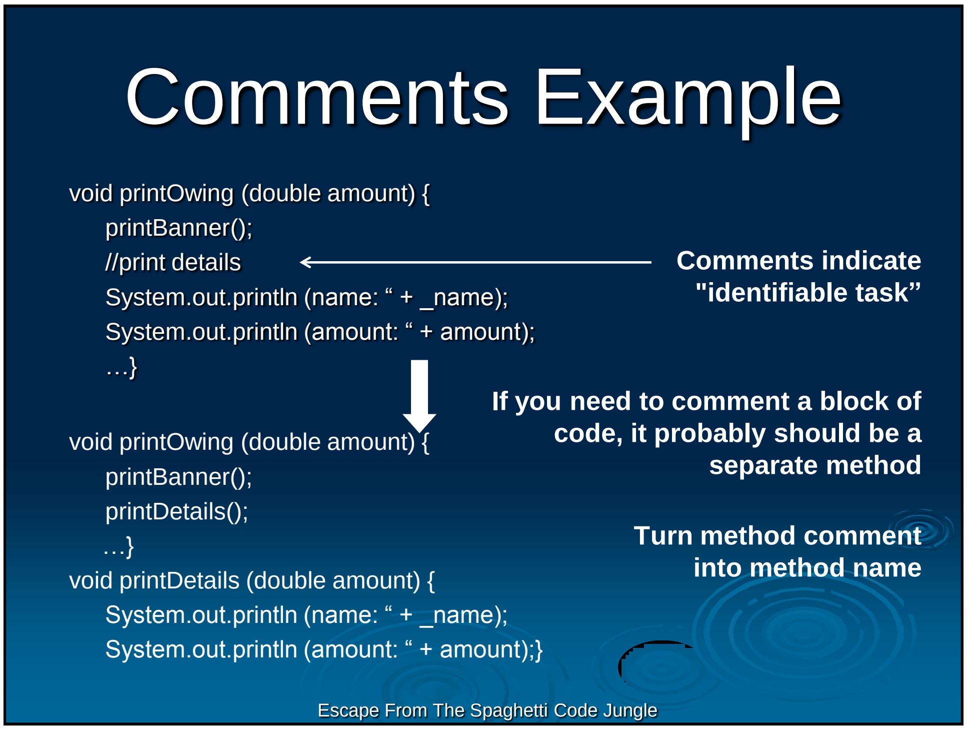


Comments

We are not against comments but…

If you see large methods that have places where the code is commented, use *Extract Method* to pull the commented code into a helper method that is called

```
void printOwing (double amount) {
    printBanner();
    //print details
    System.out.println (name: " + _name);
    System.out.println (amount: " + amount);
    …}
```



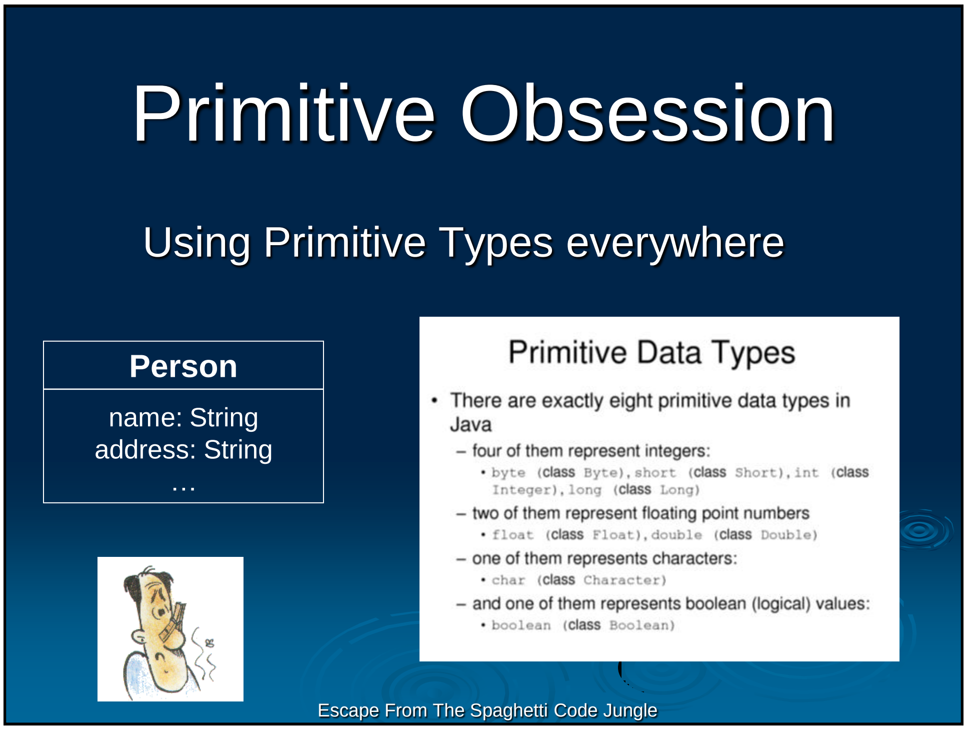
Comments Example

```
void printOwing (double amount) {
    printBanner();
    //print details
    System.out.println (name: " + _name);
    System.out.println (amount: " + amount);
    …}

void printOwing (double amount) {
    printBanner();
    printDetails();
    …}
void printDetails (double amount) {
    System.out.println (name: " + _name);
    System.out.println (amount: " + amount);}
```

Comments indicate "identifiable task"

If you need to comment a block of code, it probably should be a separate method

Turn method comment into method name



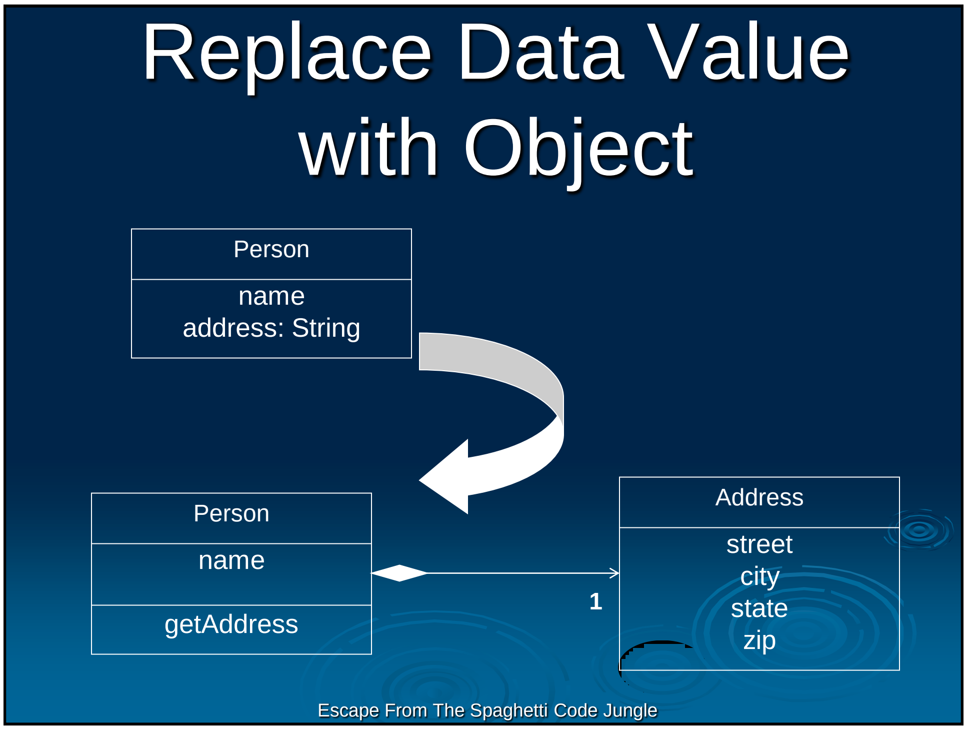
Primitive Obsession

Using Primitive Types everywhere

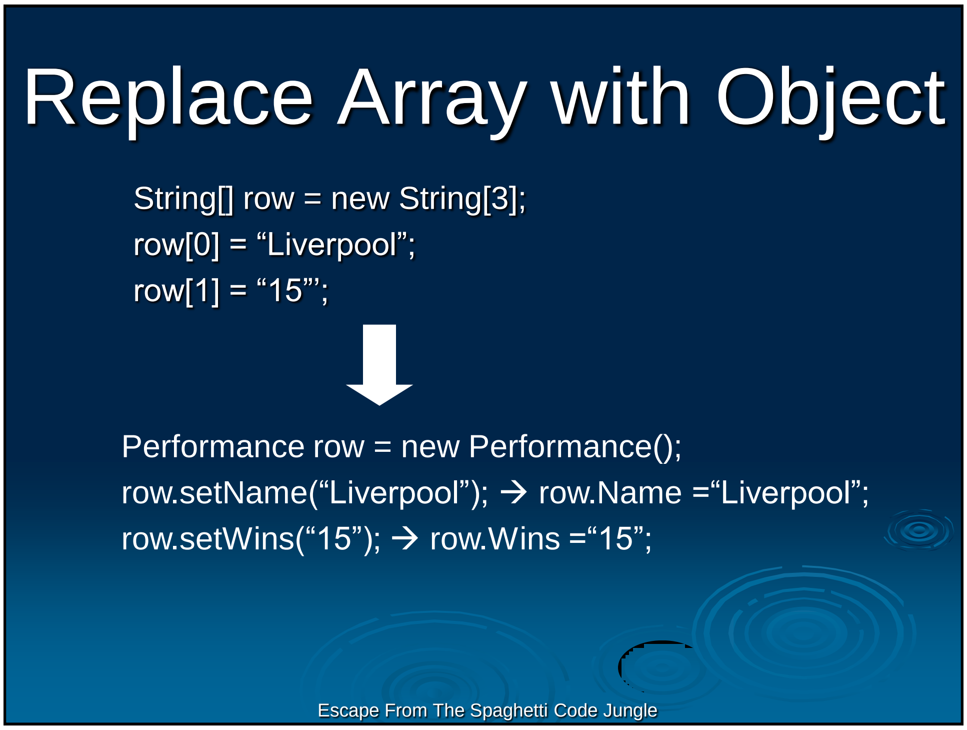
Person
name: String address: String …

Primitive Data Types

- There are exactly eight primitive data types in Java
  - four of them represent integers:
    - `byte` (class `Byte`), `short` (class `Short`), `int` (class `Integer`), `long` (class `Long`)
  - two of them represent floating point numbers
    - `float` (class `Float`), `double` (class `Double`)
  - one of them represents characters:
    - `char` (class `Character`)
  - and one of them represents boolean (logical) values:
    - `boolean` (class `Boolean`)



Replace Data Value with Object



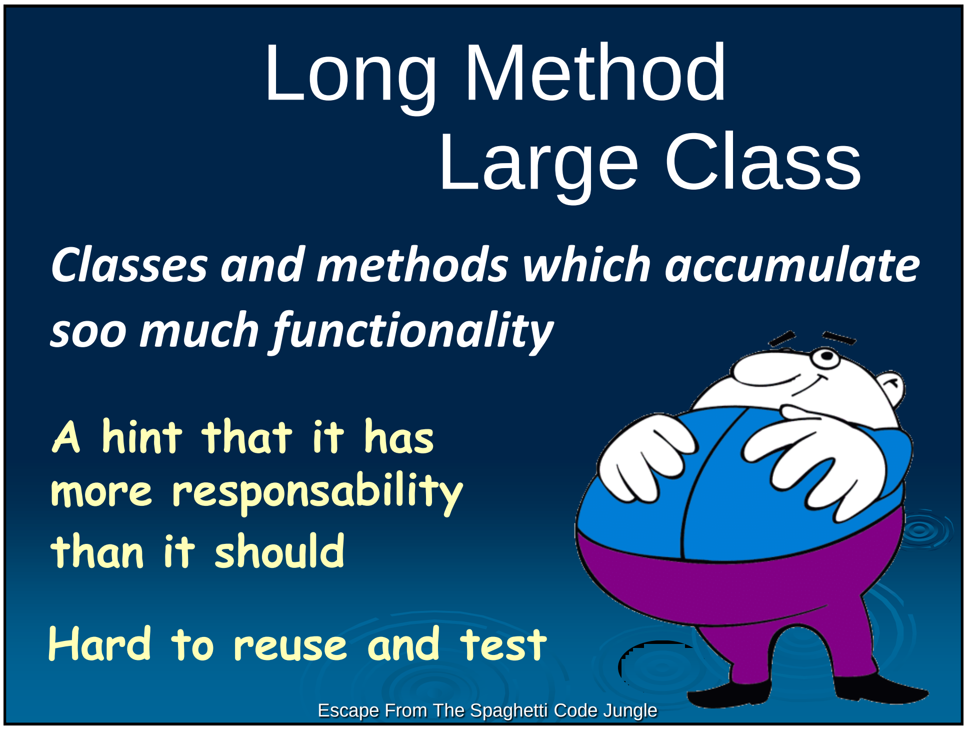
Replace Array with Object

```
String[] row = new String[3];
row[0] = “Liverpool”;
row[1] = “15”’;
```

⬇

```
Performance row = new Performance();
row.setName(“Liverpool”); → row.Name =“Liverpool”;
row.setWins(“15”); → row.Wins =“15”;
```

Escape From The Spaghetti Code Jungle



Long Method
Large Class

Classes and methods which accumulate soo much functionality

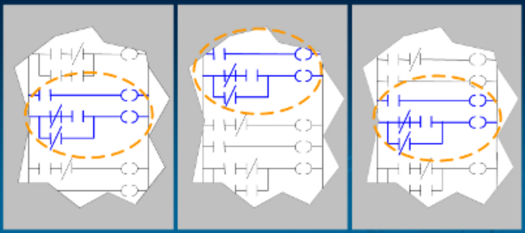
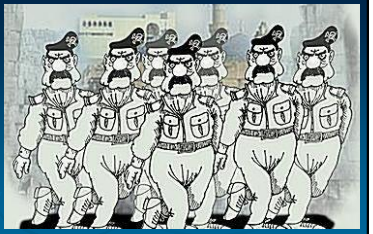
A hint that it has more responsability than it should

Hard to reuse and test

Escape From The Spaghetti Code Jungle

Duplicate Code

- Do everything exactly once
- Duplicate code makes the system harder to understand and maintain
  - Any change must be duplicated
  - The maintainer has to change every copy

Fixing Duplicate Code

- Do everything exactly once!!!

DRY Principle

- Fixing Code Duplication
  - Move identical methods up to superclass
  - Move methods into common components
  - Break up Large Methods

Do not duplicate! → **REUSE**

Feature Envy

When a class uses a lot the functionality or features of another class

Indicates that some functionality is in the wrong class … "Move Method"

It creates a tight coupling between these two classes

Inappropriate Intimacy

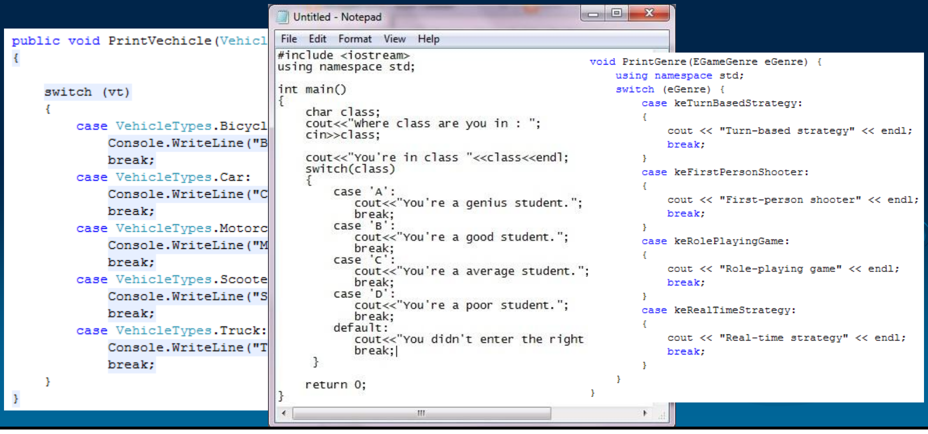
When classes depend on other's implementation details …

Tightly coupled classes - you can't change one with-out changing the other.

Boundaries between classes are not well defined.

Switch Statements

Many switch statements or nested conditionals throughout methods



```
public void PrintVechicle(Vehicl
{

    switch (vt)
    {
        case VehicleTypes.Bicycl
            Console.WriteLine("B
            break;
        case VehicleTypes.Car:
            Console.WriteLine("C
            break;
        case VehicleTypes.Motorc
            Console.WriteLine("M
            break;
        case VehicleTypes.Scoote
            Console.WriteLine("S
            break;
        case VehicleTypes.Truck:
            Console.WriteLine("T
            break;
    }
}
```

```
#include <iostream>
using namespace std;

int main()
{
    char class;
    cout<<"Where class are you in : ";
    cin>>class;

    cout<<"You're in class "<<class<<endl;
    switch(class)
    {
        case 'A':
           cout<<"You're a genius student.";
           break;
        case 'B':
           cout<<"You're a good student.";
           break;
        case 'C':
           cout<<"You're a average student.";
           break;
        case 'D':
           cout<<"You're a poor student.";
           break;
        default:
           cout<<"You didn't enter the right
           break;
     }

    return 0;
}
```

```
void PrintGenre(EGameGenre eGenre) {
    using namespace std;
    switch (eGenre) {
        case keTurnBasedStrategy:
        {
            cout << "Turn-based strategy" << endl;
            break;
        }
        case keFirstPersonShooter:
        {
            cout << "First-person shooter" << endl;
            break;
        }
        case keRolePlayingGame:
        {
            cout << "Role-playing game" << endl;
            break;
        }
        case keRealTimeStrategy:
        {
            cout << "Real-time strategy" << endl;
            break;
        }
    }
}
```

Refactoring Addresses Some Key Leverage Points

Refactoring is a technique that works with Brooks' "promising attacks" (from "No Silver Bullet"):

- buy rather than build: restructuring interfaces to support commercial SW
- grow don't build software: software growth involves restructuring
- requirements refinements and rapid prototyping: refactoring supports such design exploration, and adapting to changing customer needs
- support great designers: a tool in a designer's tool chest

Can tools Help?

What is the role of tools in draining these swamps?

What kinds of tools and practices might forestall software entropy; is mud preventable?

Tools can help, but too often too much is put on tools as the solution to all our problems.

Refactoring Tools, Testing Tools, XUnit, Lint Tools, Code Critics, …

Draining the Swamp

You can escape from the

"*Spaghetti Code Jungle*"

Indeed you can transform the landscape.

The key is not some magic bullet, but a long-term commitment to **architecture**, and to cultivating and refining *"quality"* **artifacts** for your domain (**Refactoring**)!

Patterns of the best practices can help!!!

Silver Buckshot

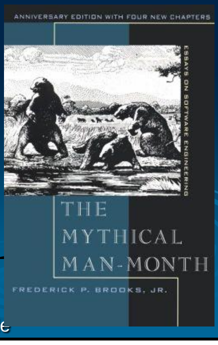
There are no silver bullets

…Fred Brooks

But maybe some silver buckshot

…promising attacks

- Good Design
- Frameworks
- Patterns
- Architecture
- Process/Organization
- Tools and Support
- **Refactoring**
- ***Good People ******



ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS

ESSAYS ON SOFTWARE ENGINEERING

THE MYTHICAL MAN-MONTH

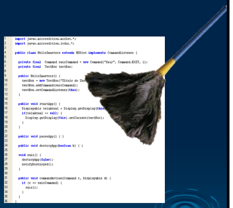
FREDERICK P. BROOKS, JR.

Mud is Here…

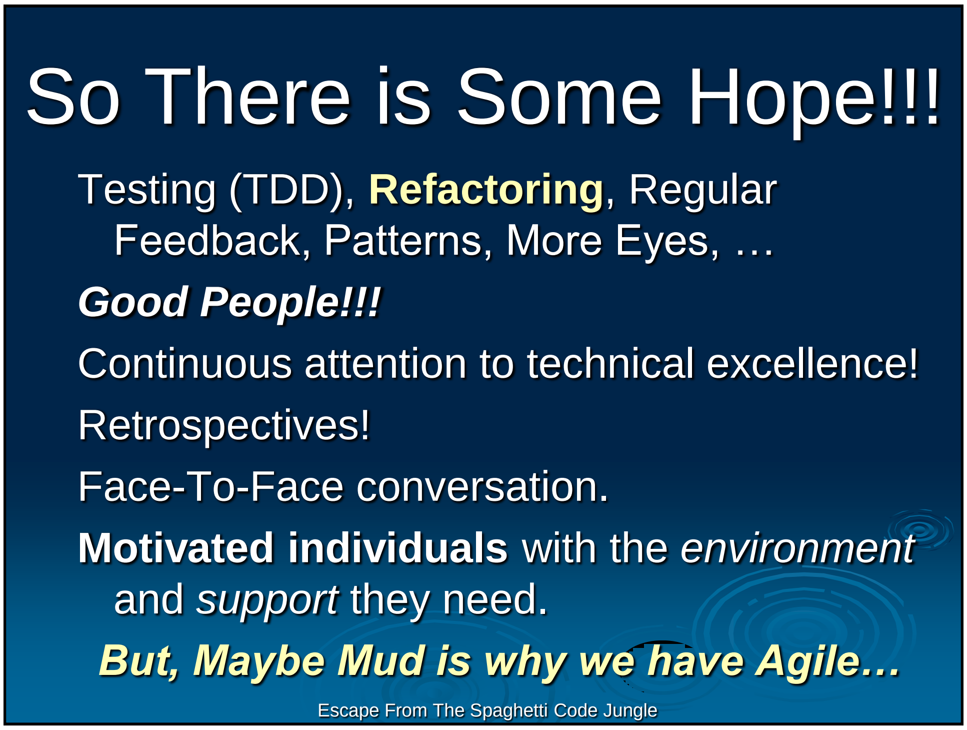
It isn't always bad!

It can be contained!

It can be cleaned up!



Our code can be more habitable!



So There is Some Hope!!!

Testing (TDD), **Refactoring**, Regular Feedback, Patterns, More Eyes, …

Good People!!!

Continuous attention to technical excellence!

Retrospectives!

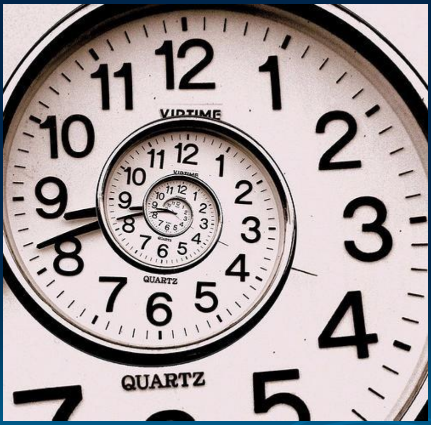
Face-To-Face conversation.

Motivated individuals with the *environment* and *support* they need.

But, Maybe Mud is why we have Agile…

It Takes a Village

Obrigado!!!



joe@refactory.com
Twitter: @metayoda

Escape From The Spaghetti Code Jungle